Curso Completo de Engenharia de Prompt: Da Fundamentação à Engenharia de Agentes

Desenvolvido por Manus, especialista em desenvolvimento de cursos

MÓDULO 1: FUNDAMENTOS DE ENGENHARIA DE PROMPT

Bem-vindo ao curso completo de Engenharia de Prompt! Neste primeiro módulo, construiremos a base sólida necessária para a sua jornada. O objetivo é desmistificar os conceitos essenciais por trás dos Modelos de Linguagem de Grande Porte (LLMs) e introduzir a disciplina que permite extrair o máximo de valor deles: a Engenharia de Prompt. Ao final deste módulo, você não apenas entenderá como essas tecnologias funcionam, mas também será capaz de construir seus primeiros prompts eficazes de forma estruturada e intencional.

1.1 Introdução ao Mundo dos LLMs

Os Modelos de Linguagem de Grande Porte, ou LLMs, representam uma das mais significativas evoluções na história da inteligência artificial. Em sua essência, um LLM é um sistema de IA treinado em vastos volumes de dados textuais, o que o capacita a compreender, gerar e interagir usando a linguagem humana de maneira notavelmente fluida. O funcionamento interno desses modelos é complexo, mas podemos entendê-lo através de três conceitos-chave: **tokens**, **contexto** e **inferência**.

- **Tokens**: São as unidades fundamentais de texto que um LLM processa. Um token pode ser uma palavra, parte de uma palavra ou até mesmo um único caractere. Por exemplo, a frase "Engenharia de Prompt" pode ser dividida em tokens como "Engenharia", "de" e "Prompt". A forma como o texto é "tokenizado" impacta diretamente como o modelo o interpreta.
- **Contexto**: Refere-se à totalidade de informações fornecidas ao modelo em uma única interação. Isso inclui não apenas a sua instrução direta, mas também exemplos, documentos de referência e o histórico da conversa. A janela de contexto (context window) é o espaço finito que um modelo pode "lembrar" em uma dada interação. Gerenciar esse espaço é um dos pilares da engenharia de contexto, que veremos mais adiante.
- **Inferência**: É o processo pelo qual o LLM, com base no contexto fornecido, gera uma resposta. O modelo calcula a probabilidade da próxima sequência de tokens que

melhor se alinha com a informação que recebeu, produzindo assim uma resposta coerente.

Em 2025, o ecossistema de LLMs é dominado por modelos altamente sofisticados como o **GPT-4o** da OpenAI, a família **Claude 4** da Anthropic e o **Gemini 1.5 Pro** do Google. Embora todos sejam extremamente capazes, eles possuem características distintas, tornando a escolha do modelo uma decisão estratégica. Por exemplo, alguns modelos podem se destacar em tarefas criativas, enquanto outros são otimizados para raciocínio lógico complexo ou para lidar com janelas de contexto extremamente longas. Compreender essas nuances é o primeiro passo para uma engenharia de prompt eficaz.

1.2 O que é Engenharia de Prompt?

A Engenharia de Prompt é a prática de projetar, refinar e otimizar os inputs (prompts) para guiar um LLM a produzir resultados específicos, precisos e de alta qualidade [1]. É a ponte entre a intenção humana e a capacidade de interpretação da máquina. Um prompt casual, como "faça um resumo", pode gerar resultados imprevisíveis. Em contrapartida, um prompt engenheirado, como "Resuma o seguinte chat de suporte ao cliente em três tópicos, focando no problema, sentimento do cliente e resolução, usando linguagem clara e concisa", transforma a ambiguidade em uma instrução acionável e com alta probabilidade de sucesso [2].

Essa disciplina é crucial para os negócios porque permite transformar LLMs de ferramentas genéricas em especialistas de domínio altamente eficientes. Empresas em setores como **jurídico**, **suporte ao cliente** e **saúde** já utilizam a engenharia de prompt para automatizar a revisão de documentos, triar tickets de suporte com precisão e auxiliar em diagnósticos, respectivamente [2].

É importante distinguir a engenharia de prompt de outras técnicas de otimização de modelos, conforme detalhado na tabela abaixo.

Técnica	Descrição
Engenharia de Prompt	Adaptar o comportamento do modelo através da formulação do input.
Fine-Tuning	Retreinar o modelo com um conjunto de dados específico do domínio.
RAG (Retrieval-Augmented Generation)	Fornecer ao modelo contexto relevante de fontes de dado externas em tempo real.

A engenharia de prompt é frequentemente o método mais rápido, acessível e econômico para melhorar o desempenho de um LLM, não exigindo infraestrutura de treinamento adicional [3].

1.3 Anatomia de um Prompt Efetivo

Um prompt bem-sucedido é composto por elementos que, juntos, eliminam a ambiguidade e fornecem ao modelo um caminho claro para a resposta desejada. Os componentes fundamentais de um prompt robusto são:

- **Instrução**: O verbo de ação que define a tarefa principal que o modelo deve executar (ex: "Analise", "Escreva", "Traduza").
- **Contexto**: Informações adicionais que o modelo precisa para executar a tarefa corretamente (ex: um artigo para resumir, dados para analisar).
- **Input**: Os dados específicos sobre os quais a ação deve ser aplicada.
- **Formato de Saída**: Uma descrição explícita de como a resposta deve ser estruturada (ex: "em formato JSON", "como uma lista de tópicos", "em uma tabela com três colunas").

O princípio mais importante é a **especificidade**. A ambiguidade é a principal causa de falhas em prompts. Um modelo não tem como adivinhar o contexto implícito na mente do usuário. Portanto, quanto mais explícito for o prompt, maior a chance de sucesso.

Exemplo de Prompt Mal Estruturado:

> "Fale sobre o novo produto."

Exemplo de Prompt Bem Estruturado:

> "Você é um especialista em marketing. Escreva um parágrafo para um post de blog (máximo de 100 palavras) anunciando nosso novo produto, o 'SyncMaster Pro'. Destaque seus três principais benefícios: integração com IA, bateria de longa duração e design premium. O tom deve ser entusiasmado e profissional."

1.4 Primeiros Passos Práticos

Agora, vamos colocar a teoria em prática. A construção de prompts eficazes começa com a aplicação de técnicas simples, mas poderosas. A técnica mais fundamental é **ser claro e direto**. Evite linguagem excessivamente complexa ou vaga. Em vez de dizer "Seria bom se você pudesse...", diga "Faça...".

Outra técnica essencial é o uso de **delimitadores**. Delimitadores, como aspas triplas (`"""`), tags XML (`<documento>...</documento>`) ou Markdown (`###`), ajudam o modelo a distinguir claramente entre diferentes partes do seu prompt, como a instrução e o contexto. Isso é

particularmente útil para evitar que o modelo confunda suas instruções com o texto que ele deve processar [4].

Finalmente, sempre **especifique o formato de saída desejado**. Se você precisa de uma lista, peça uma lista. Se precisa de um objeto JSON, forneça a estrutura exata. Isso reduz a carga de trabalho do modelo e aumenta a consistência dos resultados.

Exercício Prático:

Tarefa: Transforme o seguinte prompt vago em um prompt bem estruturado usando as técnicas aprendidas.

Prompt Vago:

> "Resuma este email. Email: [Cole o texto do email aqui]"

Pense nos seguintes pontos:

1. Qual é o objetivo do resumo?
2. Quem é o público do resumo?
3. Qual deve ser o tamanho do resumo?
4. Como o resumo deve ser formatado?

Uma possível solução será apresentada no início do próximo módulo.

Referências do Módulo 1

[1] IBM. (2025). *O que é engenharia de prompt?* Obtido de https://www.ibm.com/br-pt/think/topics/prompt-engineering [2] Lakera. (2025). *The Ultimate Guide to Prompt Engineering in 2025*. Obtido de https://www.lakera.ai/blog/prompt-engineering-guide [3] Anthropic. (2025). *Prompt engineering overview*. Obtido de https://docs.claude.com/en/docs/build-with-claude/prompt-engineering/overview [4] OpenAI. (2025). *Best practices for prompt engineering with the OpenAI API*. Obtido de https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api

---

MÓDULO 2: TÉCNICAS FUNDAMENTAIS

Solução do Exercício do Módulo 1:

Prompt Vago:

> "Resuma este email. Email: [Texto do email]"

Possível Solução Estruturada:

```
Plain Text

Resuma o email abaixo em três tópicos curtos para meu gerente. O objetivo
é informá-lo rapidamente sobre o status do projeto 'Phoenix'.

<email>
[Texto do email]
</email>

Formate a saída como uma lista de bullet points.
```

No Módulo 2, aprofundaremos nas técnicas que formam a espinha dorsal da engenharia de prompt. Estas são as ferramentas que você usará diariamente para interagir com LLMs de forma eficaz. Dominar o **Zero-Shot**, o **Few-Shot** e a **Cadeia de Pensamento (CoT)** permitirá que você resolva uma vasta gama de problemas, desde tarefas simples de classificação até desafios complexos que exigem raciocínio. Vamos explorar como dar ao modelo um papel específico e como a formatação adequada pode transformar um bom prompt em um prompt excelente.

2.1 Zero-Shot Prompting

O **Zero-Shot Prompting** é a técnica mais direta de todas. Consiste em solicitar a um LLM que execute uma tarefa sem lhe fornecer nenhum exemplo prévio de como fazê-la. O sucesso desta abordagem depende inteiramente do conhecimento pré-existente que o modelo adquiriu durante seu treinamento massivo. É como pedir a um chef experiente para criar um prato que ele nunca fez antes, confiando em sua vasta experiência culinária para deduzir o resultado esperado.

Esta técnica é ideal para tarefas genéricas e bem definidas, como tradução, sumarização de textos curtos ou classificação de sentimentos simples. A grande vantagem é a simplicidade e a rapidez, pois não requer a elaboração de exemplos.

Exemplo Prático (Classificação de Sentimento):

```
Plain Text

Classifique o sentimento do seguinte comentário como positivo, negativo ou
neutro.

Comentário: "O serviço ao cliente foi incrivelmente lento, mas o produto
em si é fantástico."

Sentimento:
```

No entanto, a limitação do Zero-Shot aparece em tarefas mais complexas ou de nicho, onde a ambiguidade pode levar a resultados incorretos. Quando a tarefa requer uma nuance que o modelo pode não ter capturado em seu treinamento, precisamos de uma abordagem mais explícita.

2.2 Few-Shot Prompting (Aprendizado por Exemplos)

Quando o Zero-Shot não é suficiente, recorremos ao **Few-Shot Prompting**. Esta técnica, também conhecida como *aprendizado no contexto* (in-context learning), consiste em fornecer ao modelo um pequeno número de exemplos (geralmente de 1 a 5) da tarefa sendo executada. Esses exemplos servem como uma demonstração, guiando o modelo a entender o padrão e o formato de saída desejado.

O poder do Few-Shot reside em sua capacidade de ensinar ao modelo uma nova tarefa "em tempo real", sem a necessidade de um custoso fine-tuning. A estrutura de um prompt Few-Shot alterna entre exemplos de input e o output correspondente, finalizando com o input para o qual se deseja a resposta.

Exemplo Prático (Extração de Dados):

> Plain Text
>
> ```
> Extraia o nome da empresa e o cargo da seguinte frase.
>
> Frase: "Após uma longa carreira, Maria se tornou a Diretora de Inovação na
> TechCorp."
> Empresa: TechCorp
> Cargo: Diretora de Inovação
>
> Frase: "João, o novo Engenheiro de Software da Innovate Solutions, começou
> hoje."
> Empresa: Innovate Solutions
> Cargo: Engenheiro de Software
>
> Frase: "A posição de Analista de Dados na DataDriven Inc. foi preenchida
> por Ana."
> Empresa:
> ```

Uma variação interessante é o **Método do Talent Show**, identificado nos materiais de referência [5]. Ele consiste em mostrar não apenas exemplos do que é bom, mas também do que deve ser evitado, refinando ainda mais a compreensão do modelo.

2.3 Chain of Thought (CoT) - Cadeia de Pensamento

Para problemas que exigem raciocínio, lógica ou múltiplos passos para serem resolvidos, a técnica de **Cadeia de Pensamento (Chain of Thought - CoT)** é revolucionária. A ideia é instruir o modelo a "pensar passo a passo", decompondo um problema complexo em etapas intermediárias antes de chegar à resposta final. Isso imita o processo de raciocínio humano e torna o caminho para a solução transparente e verificável [6].

Simplesmente adicionar a frase "Vamos pensar passo a passo" ou "Let's think step by step" ao final do prompt pode ser suficiente para ativar esse modo de raciocínio no modelo. Em uma abordagem Few-Shot, pode-se incluir os passos do raciocínio nos exemplos.

Exemplo Prático (Problema Lógico):

> Plain Text
>
> ```
> Pergunta: Uma cafeteria tem 25 mesas. 15 mesas têm 4 cadeiras cada e as
> restantes têm 2 cadeiras cada. Quantas cadeiras a cafeteria tem no total?
>
> Resposta: Vamos pensar passo a passo.
> 1. Primeiro, calculamos o número de mesas com 2 cadeiras. Total de mesas
> (25) - mesas com 4 cadeiras (15) = 10 mesas com 2 cadeiras.
> 2. Agora, calculamos o total de cadeiras nas mesas de 4 lugares. 15 mesas
> * 4 cadeiras/mesa = 60 cadeiras.
> 3. Em seguida, calculamos o total de cadeiras nas mesas de 2 lugares. 10
> mesas * 2 cadeiras/mesa = 20 cadeiras.
> 4. Finalmente, somamos os dois totais para encontrar o número total de
> cadeiras. 60 + 20 = 80 cadeiras.
>
> A resposta final é: 80.
> ```

Esta técnica melhora drasticamente o desempenho em tarefas de aritmética, bom senso e raciocínio simbólico.

2.4 Uso de Papéis e Personas (Role Prompting)

Atribuir um papel ou uma persona ao LLM é uma maneira extremamente eficaz de direcionar o tom, o estilo, o nível de detalhe e a expertise da resposta. Ao iniciar um prompt com "Você é um...", você ancora o modelo em um contexto específico, fazendo com que ele acesse o conhecimento associado àquela persona.

Essa instrução pode ser dada no **System Prompt** (uma instrução de alto nível que define o comportamento do modelo para toda a conversa) ou no início de um **User Prompt** (a instrução para uma única interação).

Exemplo Prático (Criação de Conteúdo):

> Plain Text

```
System Prompt: Você é um copywriter sênior especializado em tecnologia
B2B, com um estilo de escrita claro, conciso e focado em benefícios para o
cliente.

User Prompt: Crie um slogan para um novo software de cibersegurança
chamado 'GuardianAI', que usa inteligência artificial para prever ameaças
antes que elas aconteçam.
```

Esta técnica é simples, mas poderosa, e pode ser combinada com todas as outras para refinar ainda mais os resultados.

2.5 Formatação e Estruturação

A clareza de um prompt não depende apenas das palavras, mas também de sua estrutura visual. O uso de **tags XML** (`<exemplo>...</exemplo>`) ou **headers de Markdown** (`## Instruções`) para delinear seções ajuda o modelo a analisar e entender a estrutura do seu pedido. Essa organização é especialmente crítica em prompts longos e complexos.

O **Método de Empilhagem (Stacking Method)**, mencionado nos materiais de referência [5], refere-se à construção de prompts complexos através do empilhamento de múltiplas instruções e contextos de forma organizada. Templates reutilizáveis são uma consequência natural desta prática, permitindo a criação de prompts robustos e consistentes em escala.

Exemplo Prático (Estruturação com XML):

Plain Text

```
<instrucoes>
Resuma o artigo fornecido em 5 bullet points chave.
</instrucoes>

<artigo>
[Texto longo do artigo aqui...]
</artigo>

<formato_saida>
- Ponto 1
- Ponto 2
- Ponto 3
- Ponto 4
- Ponto 5
</formato_saida>
```

Dominar estas técnicas fundamentais lhe dará um controle sem precedentes sobre os LLMs. No próximo módulo, construiremos sobre esta base para explorar técnicas intermediárias que permitem resolver tarefas ainda mais complexas e com maior nuance.

Exercício Prático:

Tarefa: Crie um prompt que use pelo menos três das técnicas aprendidas neste módulo (ex: Few-Shot, Chain of Thought, Role Prompting) para ensinar a um LLM uma tarefa de nicho, como "traduzir gírias de gamers para uma linguagem corporativa formal".

Referências do Módulo 2

[5] Materiais do usuário, Tópico 2151. (2025). [6] Wei, J., et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Obtido de https://arxiv.org/abs/2201.11903

MÓDULO 3: TÉCNICAS INTERMEDIÁRIAS

Solução do Exercício do Módulo 2:

Tarefa: Criar um prompt que use pelo menos três técnicas (Few-Shot, CoT, Role Prompting) para traduzir gírias de gamers para linguagem corporativa.

Possível Solução Estruturada:

> Plain Text
>
> ```
> Você é um especialista em comunicação corporativa e um ávido gamer,
> perfeitamente fluente em ambos os jargões. Sua tarefa é traduzir gírias de
> games para um linguajar profissional e de fácil compreensão no ambiente de
> trabalho. Pense passo a passo para garantir que o significado e a intenção
> sejam preservados.
>
> **Exemplo 1:**
> Gíria: "Precisamos 'farmar' mais leads."
> Raciocínio: A gíria 'farmar' vem de jogos e significa realizar uma tarefa
> repetitiva para acumular recursos. No contexto de negócios, isso se traduz
> em um esforço contínuo e sistemático para gerar novos contatos.
> Tradução Corporativa: "Precisamos intensificar nossos esforços contínuos
> para a geração de leads."
>
> **Exemplo 2:**
> Gíria: "O lançamento do produto foi 'buffado'."
> Raciocínio: 'Buffar' em jogos significa fortalecer ou melhorar algo.
> Aplicado a um produto, sugere que ele recebeu melhorias significativas.
> ```

```
Tradução Corporativa: "O produto foi significativamente aprimorado em seu
último lançamento."

**Sua Tarefa:**
Gíria: "A equipe de marketing 'nerfou' a campanha no último minuto."
Raciocínio:
Tradução Corporativa:
```

Com as técnicas fundamentais dominadas, o Módulo 3 eleva o seu nível de controle sobre os LLMs. Aqui, exploraremos métodos que adicionam nuance e precisão aos seus prompts. Aprenderemos a guiar o modelo não apenas pelo que ele *deve* fazer, mas também pelo que ele *não deve* fazer. Vamos mergulhar em técnicas para decompor problemas complexos, encadear prompts para criar fluxos de trabalho e ajustar os parâmetros do modelo para refinar a qualidade da saída. Essas são as habilidades que separam um usuário casual de um verdadeiro engenheiro de prompts.

3.1 Instruções Negativas (Anti-Keyword Staining)

Enquanto a maioria das técnicas de prompt se concentra em dizer ao modelo o que fazer, as **instruções negativas** são uma ferramenta poderosa para especificar o que *evitar*. Essa abordagem, também referida como *Anti-Keyword Staining* [5], ajuda a prevenir que o modelo gere saídas indesejadas, como o uso de jargões, hashtags, emojis ou um tom de voz inadequado. É particularmente útil para garantir a conformidade com guias de estilo e para refinar a qualidade do texto gerado.

O segredo é ser explícito sobre as restrições. Em vez de apenas esperar que o modelo adivinhe, declare claramente o que deve ser omitido.

Exemplo Prático (Post para LinkedIn):

Plain Text

```
Escreva um post curto para o LinkedIn sobre a importância da
cibersegurança para pequenas empresas.

**Instruções:**
- O tom deve ser profissional e informativo.
- O texto deve ter no máximo 150 palavras.

**O que evitar:**
- Não use emojis.
- Não inclua hashtags.
- Não use gírias como "hackers do mal" ou linguagem alarmista.
```

3.2 Prefilling e Controle de Saída

A técnica de **Prefilling** (ou preenchimento prévio) é uma forma sutil, mas extremamente eficaz, de controlar o início da resposta do modelo. Ao fornecer o começo da saída desejada, você força o modelo a seguir um formato ou tom específico desde o primeiro token. Isso reduz a chance de o modelo divagar e o coloca imediatamente no caminho certo.

Essa técnica é excelente para gerar saídas estruturadas, garantir um tom de voz consistente ou forçar o modelo a começar com uma frase ou formato específico. É como dar o primeiro passo em uma dança para garantir que seu parceiro siga o ritmo correto.

Exemplo Prático (Geração de Código):

> Plain Text
>
> ```
> Escreva uma função em Python chamada `calcular_media` que recebe uma lista
> de números e retorna a média.
>
> ```python
> def calcular_media(numeros):
>   """Calcula a média de uma lista de números."""
> ```
>
> Plain Text
>
> ```
> ```

Ao fornecer o início da função e a docstring, o modelo é incentivado a completar o código de forma lógica e bem documentada.

3.3 Decomposição de Tarefas Complexas

O **Método da Decomposição** (ou *Deconstruction Method*) [5] baseia-se no princípio de "dividir para conquistar". Tarefas grandes e complexas podem sobrecarregar um LLM, levando a resultados incompletos ou de baixa qualidade. Ao decompor um problema em subtarefas menores e mais gerenciáveis, você pode guiar o modelo através de um processo lógico, garantindo que cada parte seja tratada com a devida atenção.

Essa abordagem é diferente da Cadeia de Pensamento (CoT). Enquanto a CoT pede ao modelo para *ele mesmo* detalhar os passos, a decomposição envolve o engenheiro de prompt definindo explicitamente as subtarefas a serem executadas em sequência.

Exemplo Prático (Criação de um Relatório):

> **Prompt Original (Complexo):** "Analise os dados de vendas do último trimestre, identifique as principais tendências, escreva um resumo executivo e crie três recomendações."**Prompts Decompostos:**
>
> 1. "Analise os seguintes dados de vendas do último trimestre e identifique as três principais tendências observadas. [Dados aqui]"
> 2. "Com base na análise de tendências fornecida, escreva um resumo executivo de 100 palavras. [Análise do prompt 1 aqui]"
> 3. "Com base no resumo executivo, proponha três recomendações acionáveis para aumentar as vendas no próximo trimestre. [Resumo do prompt 2 aqui]"

3.4 Encadeamento de Prompts (Chaining)

O **Método de Encadeamento** (*Chaining Method*) [5] é a implementação prática da decomposição. Ele consiste em criar um fluxo de trabalho onde a saída de um prompt se torna a entrada para o próximo. Essa técnica permite a criação de pipelines de processamento sofisticados, onde cada etapa é otimizada para uma tarefa específica. Ferramentas como LangChain e LlamaIndex são construídas em torno desse conceito.

O encadeamento é fundamental para a automação de tarefas complexas, permitindo que os LLMs realizem trabalhos que seriam impossíveis com um único prompt.

3.5 Método de Importação (Import Method)

O **Método de Importação** (*Import Method*) [5] refere-se à prática de injetar conhecimento externo e dinâmico no contexto do prompt. Em vez de depender apenas do conhecimento pré-treinado do modelo, você fornece os dados brutos ou as referências necessárias para a tarefa. Esta é a base para sistemas de RAG (Retrieval-Augmented Generation).

Essa técnica garante que as respostas sejam baseadas em informações atualizadas, precisas e específicas do domínio, superando uma das principais limitações dos LLMs: o conhecimento estático.

Exemplo Prático (Suporte Técnico):

> Plain Text
>
> ```
> Você é um agente de suporte técnico. Responda à pergunta do usuário com
> base exclusivamente no manual do produto fornecido.
>
> <manual>
> Seção 4.2: Para reiniciar o dispositivo, pressione e segure o botão de
> energia por 10 segundos.
> </manual>
> ```

```
<pergunta_usuario>
Como eu faço para reiniciar meu aparelho?
</pergunta_usuario>
```

3.6 Ajuste de Parâmetros

Além de refinar o texto do prompt, você pode controlar o comportamento do modelo ajustando seus parâmetros de inferência. Os dois parâmetros mais importantes são:

- **Temperatura**: Controla a aleatoriedade da saída. Um valor baixo (ex: 0.2) torna a resposta mais determinística e focada, ideal para tarefas factuais como extração de dados. Um valor alto (ex: 0.8) aumenta a criatividade e a diversidade, útil para brainstorming ou escrita criativa.
- **Top_p (Nucleus Sampling)**: Uma alternativa à temperatura que controla a seleção de tokens com base em sua probabilidade acumulada. Um valor de `0.9` significa que o modelo considerará apenas os tokens que compõem os 90% mais prováveis da distribuição de probabilidade.

O ajuste desses parâmetros permite um controle fino sobre o equilíbrio entre precisão e criatividade, e é uma ferramenta essencial para otimizar os resultados para casos de uso específicos.

Exercício Prático:

Tarefa: Descreva um cenário onde você usaria uma **Temperatura baixa** e outro onde usaria uma **Temperatura alta**. Crie um prompt para cada cenário que justifique sua escolha de parâmetro.

Referências do Módulo 3

[5] Materiais do usuário, Tópico 2151. (2025).

MÓDULO 4: TÉCNICAS AVANÇADAS

Solução do Exercício do Módulo 3:

Tarefa: Descrever um cenário para Temperatura baixa e um para Temperatura alta, com prompts.

Possível Solução Estruturada:

> **Cenário 1: Temperatura Baixa (ex: 0.1) Justificativa:** A tarefa é extrair informações factuais e estruturadas de um texto, onde a precisão e a consistência são cruciais. A

> criatividade é indesejada. **Prompt:**
>
> ```
> Plain Text
>
> Extraia o nome do autor, o ano de publicação e o título do artigo do
> seguinte texto. Formate a saída como um objeto JSON com as chaves "autor",
> "ano" e "título".
>
> <texto>
> O estudo seminal sobre atenção, "Attention Is All You Need", foi publicado
> por Vaswani et al. em 2017.
> </texto>
> ```

> **Cenário 2: Temperatura Alta (ex: 0.9) Justificativa:** A tarefa é gerar ideias criativas para uma campanha de marketing, onde a diversidade e a originalidade das respostas são mais importantes que a precisão factual. **Prompt:**
>
> ```
> Plain Text
>
> Você é um diretor de criação em uma agência de publicidade. Gere 5 slogans
> inovadores e cativantes para o lançamento de um novo café orgânico de
> origem única da Colômbia. O público-alvo são jovens profissionais que
> valorizam sustentabilidade e qualidade.
> ```

Bem-vindo ao Módulo 4. Você já domina as técnicas que resolvem a maioria dos desafios de prompt engineering. Agora, é hora de refinar suas habilidades e abordar os cenários mais complexos e modernos. Neste módulo, vamos explorar as nuances que diferenciam os principais modelos de IA, aprender a lidar com contextos massivos de informação e a forçar saídas estruturadas com precisão cirúrgica. Também abriremos as portas para o mundo multimodal e, finalmente, estabeleceremos um framework para testar e otimizar seus prompts de maneira sistemática, garantindo que suas soluções sejam não apenas eficazes, mas também robustas e prontas para produção.

4.1 Prompting para Diferentes Modelos

Um dos maiores equívocos na engenharia de prompt é acreditar na existência de uma "melhor prática universal". Em 2025, os modelos de ponta como **GPT-4o**, **Claude 4** e **Gemini 1.5 Pro** possuem arquiteturas, dados de treinamento e filosofias de alinhamento distintas. Essas diferenças significam que um prompt otimizado para um modelo pode não ser tão eficaz em outro [2].

- **Modelos GPT (OpenAI):** Geralmente respondem muito bem a instruções diretas e detalhadas, seguindo um estilo de "manual de instruções". A técnica de *role prompting* é particularmente eficaz.

- **Modelos Claude (Anthropic):** São conhecidos por sua forte aderência a instruções de segurança e ética. Eles se destacam em tarefas conversacionais e de escrita criativa e respondem excepcionalmente bem a prompts estruturados com tags XML. A técnica de *prefilling* e o uso de exemplos para guiar o tom são muito poderosos com Claude [3].
- **Modelos Gemini (Google):** Possuem capacidades multimodais nativas muito fortes e se beneficiam de prompts que integram texto e outros formatos de mídia. São otimizados para seguir a lógica de *Chain of Thought* e se destacam em tarefas de raciocínio complexo.

Um engenheiro de prompt avançado não se apega a um único modelo. Em vez disso, ele realiza testes comparativos para determinar qual modelo (e qual técnica de prompt) é mais adequado para a tarefa específica, considerando custo, latência e qualidade da resposta.

4.2 Long Context Tips

Com as janelas de contexto ultrapassando um milhão de tokens, a capacidade de processar documentos inteiros, bases de código ou longos históricos de conversa se tornou uma realidade. No entanto, contexto longo não é uma solução mágica. Conforme discutido na pesquisa sobre *Context Engineering*, os modelos sofrem de **"context rot"** ou deterioração de contexto: a capacidade de recordar informações diminui à medida que o volume de tokens aumenta [7].

Estudos de "agulha no palheiro" (needle-in-a-haystack) mostram que a posição da informação no contexto é crucial. Geralmente, os modelos lembram melhor das informações localizadas no **início** e no **final** do contexto. Portanto, uma estratégia avançada é colocar as instruções mais críticas ou os dados mais importantes nessas posições.

Estratégias para Contexto Longo:

1. **Posicione a instrução principal no início e repita-a (ou uma versão resumida) no final.**
2. **Se estiver resumindo um livro, por exemplo, coloque o pedido de resumo no final, após o texto completo do livro.**
3. **Use técnicas de marcação (como XML) para ajudar o modelo a navegar e localizar seções específicas dentro do texto longo.**

4.3 Structured Outputs

Para aplicações que dependem de automação, a consistência da saída é fundamental. A capacidade de forçar um LLM a gerar saídas em formatos estritos como **JSON**, **XML** ou **YAML** é uma habilidade avançada e essencial. Isso elimina a necessidade de analisar e extrair informações de texto de forma livre, o que é um processo frágil e propenso a erros.

As abordagens modernas vão além de simplesmente pedir o formato no prompt. Muitos modelos e frameworks agora oferecem suporte nativo para *structured outputs*, onde você pode fornecer um schema (como um Pydantic model em Python ou um JSON Schema) e o modelo garante que sua saída se conforme a ele.

Exemplo Prático (com JSON Schema):

> Plain Text
>
> ```
> Extraia as informações do usuário do seguinte texto e formate a saída de
> acordo com o JSON Schema fornecido.
>
> <texto>
> O pedido #1234 foi feito por Joana Silva, email joana.silva@exemplo.com.
> </texto>
>
> <json_schema>
> {
>   "type": "object",
>   "properties": {
>     "nome_completo": {"type": "string"},
>     "email": {"type": "string", "format": "email"},
>     "numero_pedido": {"type": "integer"}
>   }
> }
> </json_schema>
> ```

Ferramentas como o **LlamaExtract** da LlamaIndex automatizam esse processo, permitindo a extração de dados estruturados de documentos complexos de forma confiável [8].

4.4 Multimodal Prompting

A engenharia de prompt não se limita mais ao texto. Modelos multimodais podem processar e correlacionar informações de diferentes formatos, como imagens, áudio e vídeo. O *Multimodal Prompting* envolve a criação de prompts que combinam essas modalidades para resolver problemas mais complexos.

Exemplo Prático (Análise de Imagem):

> Plain Text
>
> ```
> [IMAGEM DE UM GRÁFICO DE VENDAS]
> ```

```
Analise a imagem do gráfico de vendas acima. Qual foi o mês de pico de
vendas e qual foi a tendência geral observada no segundo semestre?
```

As melhores práticas para prompting multimodal ainda estão evoluindo, mas algumas diretrizes incluem ser extremamente claro sobre qual parte da imagem ou do áudio analisar e combinar as instruções textuais com as referências visuais ou auditivas de forma inequívoca.

4.5 Prompt Optimization e Testing

Um prompt só é verdadeiramente bom se seu desempenho for mensurável e replicável. A otimização e o teste de prompts são o que transformam a engenharia de prompt de uma arte em uma ciência. Isso envolve a criação de um framework sistemático para avaliar a qualidade das saídas.

Passos para um Framework de Teste:

1. **Definir Métricas de Sucesso:** O que constitui uma boa resposta? (Ex: precisão factual, conformidade com o formato, ausência de alucinações).
2. **Criar um Dataset de Avaliação:** Construa um conjunto de exemplos de teste (evaluation dataset) que cubram uma variedade de cenários, incluindo casos extremos (*edge cases*).
3. **Realizar Testes A/B:** Compare o desempenho de diferentes versões de um prompt em seu dataset de avaliação para identificar qual delas produz os melhores resultados de forma consistente.
4. **Documentar e Versionar:** Trate seus prompts como código. Use um sistema de controle de versão (como o Git) para rastrear alterações e documentar por que cada mudança foi feita.

Essa abordagem disciplinada é crucial para construir sistemas de IA robustos e confiáveis, especialmente em ambientes de produção.

Exercício Prático:

Tarefa: Você está construindo um sistema para classificar e-mails de suporte em três categorias: "Técnico", "Faturamento" e "Geral". Descreva como você criaria um pequeno dataset de avaliação e como usaria o A/B testing para comparar dois prompts diferentes para essa tarefa.

Referências do Módulo 4

[2] Lakera. (2025). *The Ultimate Guide to Prompt Engineering in 2025*. [3] Anthropic. (2025). *Prompt engineering overview*. [7] Anthropic. (2025). *Effective context engineering for AI agents*. [8] LlamaIndex. (2025). *Context Engineering - What it is, and techniques to consider*.

MÓDULO 5: ENGENHARIA DE CONTEXTO

Solução do Exercício do Módulo 4:

Tarefa: Descrever como criar um dataset de avaliação e usar A/B testing para um classificador de e-mails de suporte.

Possível Solução Estruturada:

> **1. Criação do Dataset de Avaliação:** Eu criaria uma planilha com 15 a 20 e-mails de suporte reais e anonimizados. Para cada e-mail, eu incluiria:
>
> - `id_email` : Um identificador único.
> - `texto_email` : O conteúdo do e-mail.
> - `categoria_esperada` : A classificação correta ("Técnico", "Faturamento" ou "Geral"), que eu mesmo definiria como gabarito.
> - `casos_extremos` : Eu garantiria a inclusão de e-mails ambíguos, como um e-mail técnico que menciona uma fatura, para testar a robustez do prompt.
>
> **2. A/B Testing dos Prompts:** Eu definiria dois prompts para comparar:
>
> - **Prompt A (Zero-Shot Simples):** `"Classifique o seguinte e-mail de suporte como Técnico, Faturamento ou Geral. E-mail: {email_text}"`
> - **Prompt B (Few-Shot com CoT):** `"Você é um agente de suporte sênior. Classifique o e-mail a seguir como Técnico, Faturamento ou Geral. Pense passo a passo para justificar sua escolha. Forneça apenas a categoria final.\n\n[Exemplo de e-mail técnico e classificação]\n[Exemplo de e-mail de faturamento e classificação]\n\nE-mail: {email_text}"`
>
> **3. Execução e Análise:** Eu executaria ambos os prompts em todo o dataset de avaliação e compararia as saídas com a `categoria_esperada` . A métrica de sucesso seria a **acurácia** (percentual de classificações corretas). O prompt com a maior acurácia, especialmente nos casos extremos, seria o vencedor.

Até agora, focamos em como *escrever* prompts. No Módulo 5, damos um passo adiante para uma visão mais holística: a **Engenharia de Contexto**. Este campo emergente trata da arte e da ciência de gerenciar *toda* a informação que um LLM vê, não apenas as instruções. Vamos entender por que o contexto é um recurso finito e precioso, como a performance do modelo se degrada com o excesso de informação (o "context rot"), e quais técnicas podemos usar para curar e otimizar a janela de contexto. Este módulo é a ponte essencial que nos levará da engenharia de prompts para a engenharia de agentes.

5.1 Da Engenharia de Prompt à Engenharia de Contexto

A engenharia de prompt foca em encontrar as palavras e a estrutura certas para as instruções. A **Engenharia de Contexto** é a evolução natural dessa disciplina. Ela aborda uma questão mais ampla: "Qual é a configuração ideal de *toda* a informação (contexto) que maximizará a chance de o modelo se comportar como desejamos?" [7].

O contexto inclui:

- **Instruções do Sistema (System Prompt)**: Diretrizes de alto nível.
- **Ferramentas (Tools)**: As capacidades que o modelo pode usar.
- **Dados Externos**: Informações recuperadas de bancos de dados ou documentos (RAG).
- **Histórico da Mensagem**: A conversa até o momento.

Enquanto a engenharia de prompt é muitas vezes uma tarefa discreta (escrever um bom prompt), a engenharia de contexto é um processo **iterativo e de curadoria** que ocorre a cada turno de uma interação, decidindo o que entra e o que sai da janela de contexto do modelo.

5.2 Anatomia de Contexto Efetivo

O princípio fundamental da engenharia de contexto é encontrar o **menor conjunto possível de tokens de alto sinal** que leve ao resultado desejado. Menos é, muitas vezes, mais.

Isso significa que nossos *system prompts* devem estar na "altitude certa": não tão detalhados a ponto de criar uma lógica frágil e hardcoded, nem tão vagos a ponto de não fornecer orientação. Devem ser específicos o suficiente para guiar, mas flexíveis o suficiente para permitir que o modelo use suas próprias heurísticas [7]. Organizar o prompt em seções claras com delimitadores (XML, Markdown) é uma prática fundamental para ajudar o modelo a navegar pelo contexto.

5.3 Context Rot e Limitações Arquiteturais

Modelos de linguagem, como os humanos, têm uma capacidade de atenção limitada. O fenômeno do **"context rot"** (deterioração de contexto) descreve como a performance de um modelo para recordar informações diminui à medida que a janela de contexto é preenchida com mais e mais tokens. Isso não é uma falha, mas uma consequência da arquitetura Transformer, onde a atenção é distribuída entre todos os tokens [7].

Estratégias de Mitigação:

- **Posicionamento Estratégico:** Coloque as informações mais críticas no início e no final do contexto.

- **Curadoria Ativa:** Remova informações redundantes ou de baixo sinal do histórico da conversa ou dos dados recuperados.
- **Resumo:** Comprima informações menos importantes em resumos concisos.

5.4 Técnicas de Engenharia de Contexto

Com base na pesquisa de organizações como LlamaIndex e Anthropic, podemos destacar várias técnicas práticas [7, 8]:

Técnica	Descrição	Aplicaçã
Ordenação de Contexto	Organizar a informação recuperada por relevância, data ou outra métrica antes de apresentá-la ao modelo.	Em um recuper
Compressão de Contexto	Usar um LLM para resumir documentos ou conversas longas, inserindo apenas o resumo no contexto final.	Manter chat pa
Memória de Longo Prazo	Armazenar informações importantes (como fatos extraídos ou resumos de conversas) em um banco de dados vetorial para recuperação futura.	Um age com um
Informação Estruturada	Usar saídas estruturadas (JSON) como uma forma de contexto condensado para etapas futuras, em vez de passar blocos de texto.	A saída diretam texto su
Workflow Engineering	Decompor uma tarefa em uma sequência de passos (chamadas de LLM, uso de ferramentas, lógica de código), cada um com seu próprio contexto otimizado.	Um age os resul tentar f

5.5 RAG (Retrieval-Augmented Generation)

O **RAG** é talvez a aplicação mais proeminente da engenharia de contexto. É uma técnica que aprimora um LLM conectando-o a uma fonte de conhecimento externa e dinâmica. Em vez de depender apenas de seu conhecimento pré-treinado, o modelo pode "consultar" um banco de dados vetorial, uma API ou outra fonte de dados para obter informações relevantes e atualizadas em tempo real.

Arquitetura Básica de RAG:

1. **Usuário faz uma pergunta.**
2. **Sistema de Recuperação (Retriever):** A pergunta é convertida em um *embedding* (uma representação vetorial) e usada para buscar os trechos de informação mais relevantes na base de conhecimento.

3. **Aumento de Contexto (Augmentation):** Os trechos recuperados são inseridos no prompt, junto com a pergunta original do usuário.
4. **Geração (Generation):** O LLM recebe o prompt aumentado e gera uma resposta baseada tanto em seu conhecimento interno quanto nas informações fornecidas.

O RAG resolve problemas críticos como alucinações (ao basear a resposta em fatos concretos) e conhecimento desatualizado. É a espinha dorsal da maioria das aplicações de IA de nível de produção que lidam com dados privados ou em constante mudança.

Exercício Prático:

Tarefa: Imagine que você está construindo um chatbot para responder a perguntas sobre os produtos de uma empresa. O conhecimento está em vários documentos PDF. Descreva, passo a passo, como você usaria as técnicas de **Engenharia de Contexto** e **RAG** para construir este chatbot. Mencione quais técnicas você usaria e por quê.

Referências do Módulo 5

[7] Anthropic. (2025). *Effective context engineering for AI agents*. [8] LlamaIndex. (2025). *Context Engineering - What it is, and techniques to consider*.

MÓDULO 6: AGENTES DE IA - FUNDAMENTOS

Solução do Exercício do Módulo 5:

Tarefa: Descrever como construir um chatbot de Q&A sobre produtos usando Engenharia de Contexto e RAG.

Possível Solução Estruturada:

> 1. **Processamento dos Documentos (Indexing):** Primeiro, eu processaria todos os PDFs dos manuais dos produtos. Cada documento seria dividido em trechos menores (chunks) para otimizar a recuperação. Eu usaria uma biblioteca como a LlamaIndex para gerar *embeddings* para cada trecho e armazená-los em um banco de dados vetorial.
> 2. **Arquitetura RAG (Retrieval):** Quando um usuário fizesse uma pergunta, o sistema primeiro a converteria em um embedding e a usaria para buscar no banco de dados vetorial os 3-5 trechos mais relevantes dos manuais.
> 3. **Engenharia de Contexto (Augmentation):** Aqui, várias técnicas seriam aplicadas:
>   - **Método de Importação:** Os trechos recuperados seriam injetados no prompt.

> - **Ordenação de Contexto:** Os trechos seriam ordenados por relevância antes de serem adicionados ao prompt para garantir que a informação mais importante viesse primeiro.
> - **Prompt Estruturado:** Eu usaria um prompt claro com tags XML para separar a instrução, a pergunta do usuário e os documentos recuperados. Ex: `<documentos_recuperados>...</documentos_recuperados>`.
> 4. **Geração (Generation):** O LLM receberia o prompt aumentado com a instrução: `"Você é um especialista nos produtos da nossa empresa. Responda à pergunta do usuário baseando-se *exclusivamente* nas informações fornecidas nos documentos recuperados. Se a resposta não estiver nos documentos, diga que você não tem essa informação."`. Isso minimizaria alucinações.
> 5. **Memória de Longo Prazo (Opcional):** Para conversas contínuas, eu poderia usar um `VectorMemoryBlock` para armazenar o histórico da conversa, permitindo que o chatbot se lembre de perguntas anteriores e forneça um suporte mais contextual.

Bem-vindo ao Módulo 6, o ponto de inflexão do nosso curso. Até agora, tratamos o LLM como um processador de texto superavançado, respondendo a um estímulo (o prompt) com uma resposta. Agora, vamos dar ao modelo a capacidade de **agir**. Neste módulo, você aprenderá os fundamentos dos Agentes de IA: sistemas que usam LLMs como seu "cérebro" para raciocinar, planejar e interagir com o ambiente através de ferramentas. Vamos desvendar o ciclo de vida de um agente, projetar nossas primeiras ferramentas e entender os protocolos que permitem essa interação, preparando o terreno para a construção de automações verdadeiramente inteligentes.

6.1 Introdução aos Agentes de IA

Um **Agente de IA** é um sistema que utiliza um Modelo de Linguagem de Grande Porte (LLM) como seu motor de raciocínio central para interagir com um ambiente, tomar decisões e executar ações para atingir um objetivo. A principal diferença entre um LLM e um agente é que um LLM *responde*, enquanto um agente *age*.

Um agente transcende a simples geração de texto. Ele pode navegar na internet, interagir com APIs, gerenciar arquivos ou controlar outros softwares. Essa capacidade de ação é o que define os **fluxos de trabalho agênticos** (*agentic workflows*), onde tarefas complexas são decompostas e executadas de forma autônoma [9].

Os componentes essenciais de um agente são:

- **Cérebro (Brain):** Um LLM que é responsável pelo raciocínio, planejamento e tomada de decisão.
- **Percepção (Perception):** A capacidade de receber e processar informações do ambiente (ex: ler o resultado de uma API, observar uma página da web).

- **Ação (Action):** A capacidade de interagir com o ambiente através de um conjunto definido de **ferramentas (tools)**.
- **Memória (Memory):** A capacidade de reter informações de interações passadas para informar decisões futuras.

6.2 Ferramentas (Tools) para Agentes

As ferramentas são o coração da capacidade de ação de um agente. Elas são as APIs, funções ou comandos que o LLM pode invocar para interagir com o mundo exterior. O design de ferramentas eficazes é uma disciplina de engenharia de prompt por si só.

Uma boa definição de ferramenta deve incluir:

1. **Nome Claro e Conciso:** O nome da função que será chamada (ex: `pesquisar_noticias`).
2. **Descrição Detalhada:** Uma explicação em linguagem natural do que a ferramenta faz, para que o LLM saiba *quando* usá-la. Esta descrição é crucial e funciona como um prompt para a seleção da ferramenta.
3. **Schema de Input:** Os parâmetros que a ferramenta aceita, com tipos e descrições claras.
4. **Schema de Output:** O formato dos dados que a ferramenta retorna.

O processo pelo qual um LLM decide usar uma ferramenta é frequentemente chamado de **Function Calling**. O modelo, com base na pergunta do usuário e no seu objetivo, analisa as descrições das ferramentas disponíveis e, se encontrar uma correspondência, gera uma chamada de função estruturada (geralmente em JSON) que o sistema pode então executar.

Exemplo Prático (Definição de Ferramenta):

> JSON
>
> ```
> {
>   "name": "obter_previsao_tempo",
>   "description": "Obtém a previsão do tempo atual para uma cidade
> específica.",
>   "parameters": {
>     "type": "object",
>     "properties": {
>       "cidade": {
>         "type": "string",
>         "description": "A cidade e o estado, ex: São Paulo, SP"
>       }
>     },
>     "required": ["cidade"]
> ```

```
  }
}
```

6.3 Model Context Protocol (MCP)

O **Model Context Protocol (MCP)** é um padrão emergente que visa padronizar como os agentes de IA interagem com diferentes ferramentas e fontes de dados. Ele fornece uma estrutura unificada para que um agente possa descobrir, entender e utilizar as capacidades disponíveis em seu ambiente, independentemente de serem APIs locais, serviços na nuvem ou outras IAs.

O MCP abstrai a complexidade da integração, permitindo que um agente se concentre no raciocínio e na tomada de decisões, em vez de nos detalhes de implementação de cada ferramenta. Embora ainda em desenvolvimento, a compreensão de seus princípios é fundamental para construir sistemas de agentes interoperáveis e escaláveis.

6.4 Memória e Estado em Agentes

A memória é o que permite que um agente aprenda com suas experiências e mantenha a continuidade ao longo do tempo. Podemos dividir a memória de um agente em duas categorias principais:

- **Memória de Curto Prazo (Working Memory):** É o histórico da conversa e as observações recentes, tudo contido na janela de contexto do LLM. É volátil e limitada pelo tamanho do contexto.
- **Memória de Longo Prazo (Persistent Memory):** É um armazenamento externo (como um banco de dados vetorial ou um banco de dados de grafos) onde o agente pode salvar e recuperar informações importantes, como fatos aprendidos, resumos de interações passadas ou preferências do usuário. A engenharia de contexto, que vimos no módulo anterior, é a chave para gerenciar a interação entre a memória de curto e longo prazo.

O gerenciamento de estado é o processo de rastrear o progresso do agente em direção ao seu objetivo, incluindo as ações que ele já tomou e os resultados que observou.

6.5 Loops Agênticos

O comportamento de um agente é definido por um ciclo contínuo, conhecido como **loop agêntico**. O loop mais comum é o **ReAct (Reason + Act)**, que segue este padrão:

1. **Raciocínio (Reason):** Com base no objetivo e nas observações, o LLM "pensa" sobre o que fazer a seguir. Ele pode decompor o problema, formular uma estratégia e decidir qual ferramenta usar.

2. **Ação (Act):** O LLM gera a chamada para a ferramenta escolhida. O sistema executa a ferramenta e captura o resultado.
3. **Observação (Observation):** O resultado da ação (a observação) é inserido de volta no contexto do LLM.
4. **Repetição:** O ciclo recomeça, com o LLM agora raciocinando com base na nova observação, até que o objetivo seja alcançado ou uma condição de parada seja atingida.

Controlar esses loops, definir condições de parada claras e lidar com erros (como uma falha de API) são desafios centrais na construção de agentes robustos.

Exercício Prático:

Tarefa: Você quer construir um agente simples para responder à pergunta: "Qual é a capital da França e qual é a sua população atual?". Descreva o provável **loop agêntico (ReAct)** que o agente seguiria. Quais ferramentas ele precisaria? Quais seriam os passos de raciocínio, ação e observação?

Referências do Módulo 6

[9] IBM. (2025). *What are Agentic Workflows?*. Obtido de https://www.ibm.com/think/topics/agentic-workflows

---

MÓDULO 7: AGENTES AVANÇADOS E SKILLS

Solução do Exercício do Módulo 6:

Tarefa: Descrever o loop agêntico para responder "Qual é a capital da França e qual é a sua população atual?".

Possível Solução Estruturada:

> **Ferramentas Necessárias:**
>
> 1. `search(query: str)`: Uma ferramenta para pesquisar informações na web.
>
> **Loop Agêntico (ReAct):**
>
> - **Turno 1:**
> - **Raciocínio:** O usuário tem duas perguntas: a capital da França e a população dessa capital. Eu não sei a população. Preciso primeiro descobrir a capital e depois pesquisar a população dela.
> - **Ação:** Chamar `search(query="capital da França")`.
> - **Observação:** A busca retorna "Paris".

> - **Turno 2:**
>   - **Raciocínio:** Agora eu sei que a capital é Paris. Preciso encontrar a população de Paris.
>   - **Ação:** Chamar `search(query="população atual de Paris")`.
>   - **Observação:** A busca retorna "Aproximadamente 2.1 milhões de habitantes em 2025".
> - **Turno 3:**
>   - **Raciocínio:** Eu tenho todas as informações necessárias para responder à pergunta do usuário.
>   - **Ação:** Gerar a resposta final.
>   - **Observação:** "A capital da França é Paris, e sua população atual é de aproximadamente 2.1 milhões de habitantes."

Seja bem-vindo ao Módulo 7, onde a teoria dos agentes se transforma em aplicação de ponta. Com os fundamentos estabelecidos, agora vamos explorar as implementações mais sofisticadas e proprietárias que definem o estado da arte em 2025. Mergulharemos fundo nas **Claude Skills** da Anthropic, uma arquitetura poderosa para estender as capacidades dos agentes. Vamos dominar o **Extended Thinking** para resolver problemas mais complexos e, finalmente, aplicaremos todo esse conhecimento na construção de agentes de automação e de voz, culminando na exploração de sistemas multi-agente.

7.1 Claude Skills (Agent Skills)

Em outubro de 2025, a Anthropic introduziu as **Skills**, uma arquitetura inovadora para estender as capacidades do Claude e torná-lo um agente mais especializado e eficiente. Uma *Skill* é, em essência, uma pasta que contém um conjunto de instruções, scripts e recursos que o Claude pode carregar e utilizar dinamicamente quando detecta que são relevantes para uma tarefa [10].

As Skills são definidas por quatro características principais:

- **Composabilidade:** Múltiplas Skills podem ser combinadas. O Claude identifica e coordena automaticamente o uso de várias Skills para resolver um problema complexo.
- **Portabilidade:** Uma Skill é construída uma vez e pode ser usada em todos os produtos Claude (API, Apps, Claude Code).
- **Eficiência:** O agente carrega apenas as informações e arquivos mínimos necessários da Skill, mantendo a velocidade e a eficiência de tokens.
- **Poder:** As Skills podem incluir código executável, permitindo que tarefas determinísticas (como cálculos complexos ou manipulação de arquivos) sejam

executadas com a confiabilidade da programação tradicional, em vez da geração de tokens probabilística.

Para desenvolvedores, as Skills são gerenciadas através de uma API (`/v1/skills`) e requerem o uso do *Code Execution Tool*, um ambiente seguro para a execução de código. A própria Anthropic fornece Skills para tarefas comuns, como a geração de planilhas Excel, apresentações PowerPoint e documentos Word, mas o verdadeiro poder reside na capacidade dos desenvolvedores de criar Skills customizadas para seus próprios fluxos de trabalho [10].

7.2 Extended Thinking

O **Extended Thinking** é outra inovação da Anthropic, projetada para melhorar a qualidade do raciocínio do Claude em problemas que exigem uma análise mais profunda. Em vez de gerar uma resposta imediatamente, o modelo entra em um modo de "pensamento visível", onde ele externaliza sua cadeia de raciocínio, analisa o problema de diferentes ângulos, considera e descarta hipóteses, e só então sintetiza uma resposta final [11].

Essa técnica é particularmente útil para:

- Perguntas complexas ou ambíguas que não têm uma resposta direta.
- Tarefas que envolvem criatividade e brainstorming.
- Cenários onde a transparência do processo de raciocínio é importante.

O *Think Tool*, introduzido pela Anthropic, é uma implementação prática disso, permitindo que o modelo faça uma pausa e "pense" dentro de tags `<thinking>...</thinking>` antes de prosseguir com uma ação ou resposta. Para um engenheiro de prompt, isso significa que podemos instruir explicitamente o modelo a usar esse modo de pensamento para problemas mais difíceis, garantindo uma análise mais robusta.

7.3 Agentes de Automação

Um agente de automação é um agente de IA projetado para executar fluxos de trabalho de negócios de ponta a ponta. Ele combina o raciocínio de um LLM com a capacidade de interagir com sistemas de software do mundo real, como CRMs, ERPs, bancos de dados e APIs de terceiros. Eles são a evolução natural da Automação de Processos Robóticos (RPA), mas com uma capacidade de adaptação e tomada de decisão muito superior.

Casos de Uso Comuns:

- **Processamento de Faturas:** Um agente que recebe uma fatura por e-mail, usa uma ferramenta de OCR para extrair os dados, valida as informações em um sistema financeiro e agenda o pagamento.

- **Onboarding de Clientes:** Um agente que interage com um novo cliente por chat, coleta as informações necessárias, cria uma conta no CRM e envia um e-mail de boas-vindas personalizado.
- **Monitoramento de Sistemas:** Um agente que observa logs de um sistema, detecta anomalias usando seu raciocínio, e cria um ticket de incidente no Jira com uma análise preliminar.

Construir agentes de automação robustos requer um excelente design de ferramentas, um gerenciamento de estado impecável e um tratamento de erros sofisticado.

7.4 Agentes de Voz (Voice AI Agents)

Os agentes de voz representam uma das fronteiras mais empolgantes da IA. Eles combinam um LLM com tecnologias de **reconhecimento de fala (Speech-to-Text)** e **síntese de voz (Text-to-Speech)** para criar experiências conversacionais em tempo real. Plataformas como ElevenLabs, Voiceflow e Google Dialogflow estão na vanguarda dessa tecnologia [12].

O principal desafio técnico para agentes de voz é a **latência**. Para uma conversa parecer natural, a latência de ponta a ponta (do final da fala do usuário ao início da resposta da IA) precisa ser de sub-100 milissegundos. Isso requer uma otimização extrema de cada componente da arquitetura.

Casos de Uso:

- **Atendimento ao Cliente por Telefone:** Agentes que podem atender chamadas, entender a intenção do cliente, acessar informações de sistemas e resolver problemas complexos de forma autônoma.
- **Qualificação de Leads de Vendas:** Agentes que podem ligar para potenciais clientes, fazer perguntas de qualificação e agendar uma reunião com um vendedor humano.
- **Assistentes Pessoais:** Agentes de voz que podem gerenciar calendários, fazer reservas e executar tarefas em nome do usuário.

7.5 Sistemas Multi-Agente

Um sistema multi-agente é uma arquitetura onde múltiplos agentes de IA colaboram para resolver um problema que seria muito complexo para um único agente. Nesse modelo, cada agente pode ter uma especialização ou um papel diferente, funcionando como uma equipe de especialistas humanos.

Padrões Comuns de Arquitetura:

- **Debate:** Dois ou mais agentes debatem um tópico de diferentes perspectivas para chegar a uma solução mais robusta.

- **Hierarquia:** Um agente "gerente" decompõe uma tarefa e delega subtarefas a agentes "trabalhadores" especializados.
- **Pipeline:** Um agente executa a primeira etapa de uma tarefa e passa seu resultado para o próximo agente na linha, e assim por diante.

Por exemplo, para escrever um relatório de pesquisa de mercado, um agente "pesquisador" poderia coletar dados da web, um agente "analista" poderia analisar os dados e identificar tendências, e um agente "escritor" poderia sintetizar tudo em um relatório bem escrito. A coordenação e a comunicação entre os agentes são os principais desafios de engenharia nesses sistemas.

Exercício Prático:

Tarefa: Imagine que você foi encarregado de criar uma **Skill** para o Claude que ajuda a planejar viagens. A Skill deve ser capaz de encontrar voos e reservar hotéis. Descreva a estrutura de pastas e arquivos que essa Skill poderia ter e quais informações você colocaria no arquivo de instruções principal (`SKILL.md`).

Referências do Módulo 7

[10] Anthropic. (2025, October 16). *Claude Skills: Customize AI for your workflows*. [11] Anthropic. (2025, February 24). *Claude's extended thinking*. [12] ElevenLabs. (2025). *Conversational AI Agent Platform for Real-Time Voice & Chat*.

MÓDULO 8: MASTERCLASSES

Solução do Exercício do Módulo 7:

Tarefa: Descrever a estrutura de uma Skill do Claude para planejamento de viagens.

Possível Solução Estruturada:

> **Estrutura de Pastas e Arquivos:**
>
> Plain Text
>
> ```
> /travel_planner_skill
> ├── SKILL.md
> ├── /tools
> │   ├── find_flights.py
> │   └── book_hotel.py
> └── /resources
>     └── airport_codes.csv
> ```

Conteúdo do `SKILL.md`:

Markdown

```
# Skill: Planejador de Viagens

## Descrição
Esta Skill ajuda os usuários a planejar viagens completas, encontrando os
melhores voos e reservando acomodações.

## Capacidades
- **Busca de Voos:** Utiliza a ferramenta `find_flights` para pesquisar
voos com base na origem, destino e datas.
- **Reserva de Hotéis:** Utiliza a ferramenta `book_hotel` para encontrar
e reservar hotéis na cidade de destino para as datas especificadas.

## Instruções de Uso
Ao usar esta Skill, sempre siga estes passos:
1.  Primeiro, confirme com o usuário as cidades de origem e destino, bem
como as datas de partida e retorno.
2.  Use a ferramenta `find_flights` para encontrar as opções de voos.
Apresente as 3 melhores opções ao usuário.
3.  Após o usuário selecionar um voo, use a ferramenta `book_hotel` para
encontrar acomodações. Confirme o tipo de quarto e o orçamento com o
usuário antes de reservar.
4.  Sempre peça confirmação final antes de realizar qualquer reserva.
```

Bem-vindo ao Módulo 8, o ápice da sua jornada em engenharia de prompt. As Masterclasses são projetadas para levá-lo do conhecimento teórico à aplicação em nível de especialista. Aqui, não se trata mais de aprender novas técnicas isoladas, mas de sintetizar tudo o que você aprendeu para resolver problemas complexos do mundo real. Cada masterclass é um mergulho profundo em uma área crítica, combinando estratégia, arquitetura e execução, e culminando em um projeto prático que solidificará sua expertise e construirá seu portfólio como um engenheiro de agentes de IA de elite.

Masterclass 1: Engenharia de Contexto Avançada

Objetivo: Dominar a arte de gerenciar o contexto em sistemas de IA complexos e de longa duração, otimizando a performance, reduzindo custos e garantindo a relevância da informação.

Nesta masterclass, vamos além do RAG básico. Exploraremos arquiteturas sofisticadas para curadoria de contexto em tempo real. Você aprenderá a diagnosticar e resolver problemas de *context rot*, a implementar estratégias de compressão de contexto e a projetar sistemas de memória híbridos que combinam o melhor da memória de curto e longo prazo. O foco é

a eficiência e a escalabilidade, garantindo que seus agentes permaneçam inteligentes e responsivos, mesmo ao lidar com volumes massivos de informação.

Tópicos Principais:

- Arquiteturas avançadas de RAG (Re-ranking, Multi-Query, Small-to-Big).
- Estratégias de compressão de contexto (Summarization, Fact Extraction).
- Técnicas de depuração para problemas de atenção e recuperação de informação.
- Design de sistemas de memória híbrida (vetorial, grafo, relacional).
- Otimização de custo e latência em pipelines de contexto.

Projeto Prático: Construir um agente de pesquisa que monitora continuamente um feed de notícias sobre um determinado setor. O agente deve ser capaz de manter um "estado de conhecimento" atualizado, responder a perguntas complexas que exijam a síntese de múltiplas fontes ao longo do tempo e descartar informações desatualizadas ou irrelevantes de sua memória de trabalho.

Masterclass 2: Prompt Engineering para Produção

Objetivo: Aprender a levar prompts e sistemas agênticos do ambiente de prototipação para um ambiente de produção robusto, seguro e monitorado.

Um prompt que funciona bem no playground pode falhar espetacularmente em produção. Esta masterclass foca na disciplina de engenharia de software aplicada à engenharia de prompt. Abordaremos o ciclo de vida completo do desenvolvimento de prompts, desde o versionamento e testes automatizados (CI/CD) até o monitoramento da performance em tempo real. Um foco especial será dado à segurança: como defender seus sistemas contra ataques como *prompt injection* e *jailbreaking*, e como implementar barreiras de proteção (*guardrails*) eficazes.

Tópicos Principais:

- Versionamento de prompts e datasets de avaliação com Git.
- Frameworks de avaliação de prompts (Promptfoo, LangSmith).
- CI/CD para prompts: testes automatizados de regressão.
- Monitoramento e observabilidade: rastreamento de latência, custo e qualidade da resposta.
- Segurança: defesa contra prompt injection, vazamento de dados e jailbreaking.
- Estratégias de mitigação de riscos e implementação de *guardrails*.

Projeto Prático: Pegar um prompt desenvolvido em um módulo anterior, criar um dataset de avaliação robusto para ele (incluindo testes de segurança), integrar o prompt em um

pipeline de CI/CD (usando GitHub Actions, por exemplo) que o testa automaticamente, e configurar um dashboard simples de monitoramento para rastrear sua performance.

Masterclass 3: Construindo Agentes Robustos

Objetivo: Projetar e construir agentes de IA que sejam confiáveis, resilientes a falhas e capazes de lidar com a imprevisibilidade do mundo real.

Agentes em produção falham. APIs ficam indisponíveis, ferramentas retornam resultados inesperados e o LLM pode entrar em loops ou alucinar. Um agente robusto é aquele que antecipa essas falhas e sabe como se recuperar. Nesta masterclass, focaremos em padrões de arquitetura para tratamento de erros, novas tentativas (*retries*) e validação de estado. Você aprenderá a construir agentes que não apenas executam tarefas, mas o fazem de forma confiável e previsível.

Tópicos Principais:

- Padrões de tratamento de erros em loops agênticos.
- Estratégias de *retry* com *backoff* exponencial para chamadas de API.
- Validação de saída de ferramentas e do raciocínio do LLM.
- Técnicas para detectar e quebrar loops de raciocínio.
- Arquiteturas de agentes stateful vs. stateless.
- Estudo de casos de falhas de agentes em produção e como foram resolvidas.

Projeto Prático: Construir um agente que automatiza um processo de reserva em um site de viagens de teste. O projeto deve focar explicitamente no tratamento de erros. O agente deve ser capaz de lidar com elementos de página que não carregam, timeouts de API, mensagens de erro inesperadas do site e se recuperar de forma inteligente para completar a reserva.

Masterclass 4: O Futuro da Engenharia de Agentes

Objetivo: Explorar as tendências emergentes que estão moldando o futuro da interação humano-IA e preparar-se para a próxima geração de sistemas agênticos.

O campo da IA está se movendo a uma velocidade vertiginosa. Esta masterclass é uma visão estratégica do que está por vir. Discutiremos as tendências mais recentes em pesquisa de IA, como modelos de ação, sistemas auto-aperfeiçoáveis e a convergência de visão, linguagem e robótica. Também debateremos o impacto da IA agêntica no futuro do trabalho e as considerações éticas que todo engenheiro de IA deve ter em mente. O objetivo é fornecer um mapa para sua evolução contínua como um profissional na vanguarda da tecnologia.

Tópicos Principais:

- Modelos de Ação e o fim do "Function Calling" explícito.
- Agentes auto-aperfeiçoáveis e aprendizado por reforço a partir de feedback humano (RLHF).
- Sistemas multi-agente e inteligência de enxame (swarm intelligence).
- O impacto da IA agêntica no mercado de trabalho e na automação do conhecimento.
- Ética em IA: responsabilidade, transparência e alinhamento de agentes autônomos.
- Como se manter atualizado e continuar aprendendo em um campo em constante mudança.

Atividade: Sessão de debate e brainstorming em grupo sobre um desafio social ou de negócios complexo (ex: otimização da cadeia de suprimentos global, combate à desinformação). Os grupos deverão projetar uma arquitetura de sistema multi-agente para abordar o problema, apresentando suas ideias e defendendo suas escolhas de design, focando nas implicações éticas e de longo prazo.

REFERÊNCIAS COMPLETAS DO CURSO

[1] IBM. (2025). *O que é engenharia de prompt?* Obtido de https://www.ibm.com/br-pt/think/topics/prompt-engineering

[2] Lakera. (2025). *The Ultimate Guide to Prompt Engineering in 2025*. Obtido de https://www.lakera.ai/blog/prompt-engineering-guide

[3] Anthropic. (2025). *Prompt engineering overview*. Obtido de https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/overview

[4] OpenAI. (2025). *Best practices for prompt engineering with the OpenAI API*. Obtido de https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api

[5] Materiais do usuário, Tópico 2151. (2025). Transcrição de palestra sobre métodos avançados de Prompt Engineering.

[6] Wei, J., et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Obtido de https://arxiv.org/abs/2201.11903

[7] Anthropic. (2025). *Effective context engineering for AI agents*. Obtido de https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

[8] LlamaIndex. (2025). *Context Engineering - What it is, and techniques to consider*. Obtido de https://www.llamaindex.ai/blog/context-engineering-what-it-is-and-techniques-to-consider

[9] IBM. (2025). *What are Agentic Workflows?* Obtido de https://www.ibm.com/think/topics/agentic-workflows

[10] Anthropic. (2025, October 16). *Claude Skills: Customize AI for your workflows*. Obtido de https://www.anthropic.com/news/skills

[11] Anthropic. (2025, February 24). *Claude's extended thinking*. Obtido de https://www.anthropic.com/news/extended-thinking

[12] ElevenLabs. (2025). *Conversational AI Agent Platform for Real-Time Voice & Chat*. Obtido de https://elevenlabs.io/conversational-ai

---

GLOSSÁRIO DE TERMOS

Agentic Workflow: Fluxo de trabalho onde um agente de IA executa tarefas de forma autônoma, tomando decisões e interagindo com ferramentas.

Chain of Thought (CoT): Técnica de prompting que instrui o modelo a externalizar seu raciocínio passo a passo antes de chegar a uma resposta final.

Chunking: Processo de dividir documentos longos em trechos menores e semanticamente coesos para otimizar a recuperação em sistemas RAG.

Context Rot: Fenômeno onde a capacidade de um LLM de recordar informações diminui à medida que a janela de contexto é preenchida.

Context Window: O espaço finito de tokens que um LLM pode processar em uma única interação.

Embedding: Representação vetorial de um texto que captura seu significado semântico, usada para busca por similaridade.

Few-Shot Prompting: Técnica que fornece ao modelo alguns exemplos da tarefa antes de solicitar a execução.

Fine-Tuning: Processo de retreinar um modelo pré-treinado com dados específicos de um domínio.

Function Calling: Capacidade de um LLM de gerar chamadas estruturadas para ferramentas externas.

Guardrails: Barreiras de proteção implementadas para prevenir comportamentos indesejados ou inseguros de um LLM.

Hallucination: Quando um LLM gera informações que parecem plausíveis, mas são factualmente incorretas ou inventadas.

In-Context Learning: Capacidade de um LLM de aprender uma nova tarefa a partir de exemplos fornecidos no contexto, sem retreinamento.

Jailbreaking: Tentativa de contornar as restrições de segurança de um LLM através de prompts manipulados.

LLM (Large Language Model): Modelo de IA treinado em grandes volumes de texto para compreender e gerar linguagem humana.

MCP (Model Context Protocol): Padrão para padronizar a interação entre agentes de IA e ferramentas/fontes de dados.

Multimodal: Capacidade de processar e gerar múltiplos tipos de dados (texto, imagem, áudio, vídeo).

Prefilling: Técnica de fornecer o início da resposta desejada para guiar o formato e tom da saída do modelo.

Prompt Injection: Ataque onde instruções maliciosas são inseridas no input para manipular o comportamento do LLM.

RAG (Retrieval-Augmented Generation): Técnica que aprimora um LLM conectando-o a fontes de conhecimento externas para recuperação de informações em tempo real.

ReAct (Reason + Act): Padrão de loop agêntico que alterna entre raciocínio, ação e observação.

Role Prompting: Técnica de atribuir uma persona ou papel específico ao LLM para direcionar seu comportamento.

Schema: Estrutura que define o formato esperado de dados, como JSON Schema.

Skill (Claude Skills): Arquitetura da Anthropic que permite estender as capacidades do Claude através de pastas contendo instruções, scripts e recursos.

System Prompt: Instrução de alto nível que define o comportamento geral do modelo para toda uma conversa.

Temperature: Parâmetro que controla a aleatoriedade da saída de um LLM (baixo = determinístico, alto = criativo).

Token: Unidade básica de texto processada por um LLM (pode ser uma palavra, parte de palavra ou caractere).

Top_p (Nucleus Sampling): Parâmetro que controla a diversidade da saída limitando a seleção de tokens por probabilidade acumulada.

Vector Database: Banco de dados otimizado para armazenar e buscar embeddings vetoriais por similaridade.

Zero-Shot Prompting: Técnica de solicitar a um LLM que execute uma tarefa sem fornecer exemplos prévios.

FIM DO CONTEÚDO DO CURSO

Este curso foi desenvolvido com dedicação para fornecer uma formação completa e prática em Engenharia de Prompt. Esperamos que você aproveite cada módulo e se torne um especialista na arte e ciência de interagir com LLMs e construir agentes de IA robustos.

Desenvolvido por Manus | Versão 1.0 - Outubro 2025